Week 12 : Visualisation (continued)

pending tasks

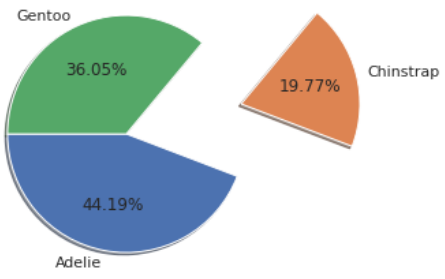
type

Pie charts

- The composition of data can be static or dynamic.
- Pie plots, stacked bar plots are an example of static composition of data.

1. Example using penguins dataset : counting the number of penguins across species.
    - Explode is used to remove a piece of pie and the start angle can also be changed.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
# counting the number of distinct species in the penguin dataset.
c = p.groupby('species')['species'].count()
plt.pie(c, labels=c.index, autopct="%.2f%%",
        explode=[0, 1, 0], startangle=180, shadow=True);
plt.show()
```



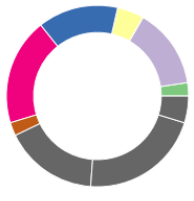
2. The association of the number to the chart is difficult as the fraction corresponds to the angle but the area makes it difficult to understand this. Also, if there are too many composing units, it becomes difficult to visualise.

Donut chart

- Donut plots can be plotted using the wedgeprop parameter in the pie.
- It gives a better visual representation than pie charts, where the area in the segment is misleading.
- Colormaps can be found and used from matplotlib's colormap. Qualitative colormaps are good for pie and donut charts.

1. Example syntax of donut chart.

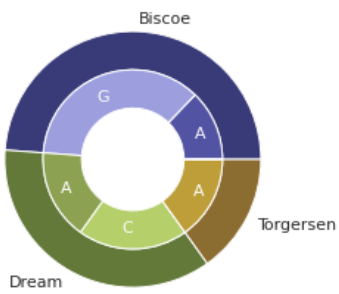
```
#setting the colour map
cmap = plt.get_cmap('Accent')
my_colours = cmap(np.arange(10))
# drawing the donut chart
plt.pie(np.random.randint(0, 10, 10),
        wedgeprops=dict(width=0.3),
        colors=my_colours);
plt.show()
```



2. Example using penguins data.

- The empty space in the middle can be used to add another level of composition. This data can be obtained using pandas crosstab. The index corresponds to the outer ring while the inner ring corresponds to the column.i.e each row represents the inner donut value.
- The colours in the outer ring must correspond to those in the inner ring.

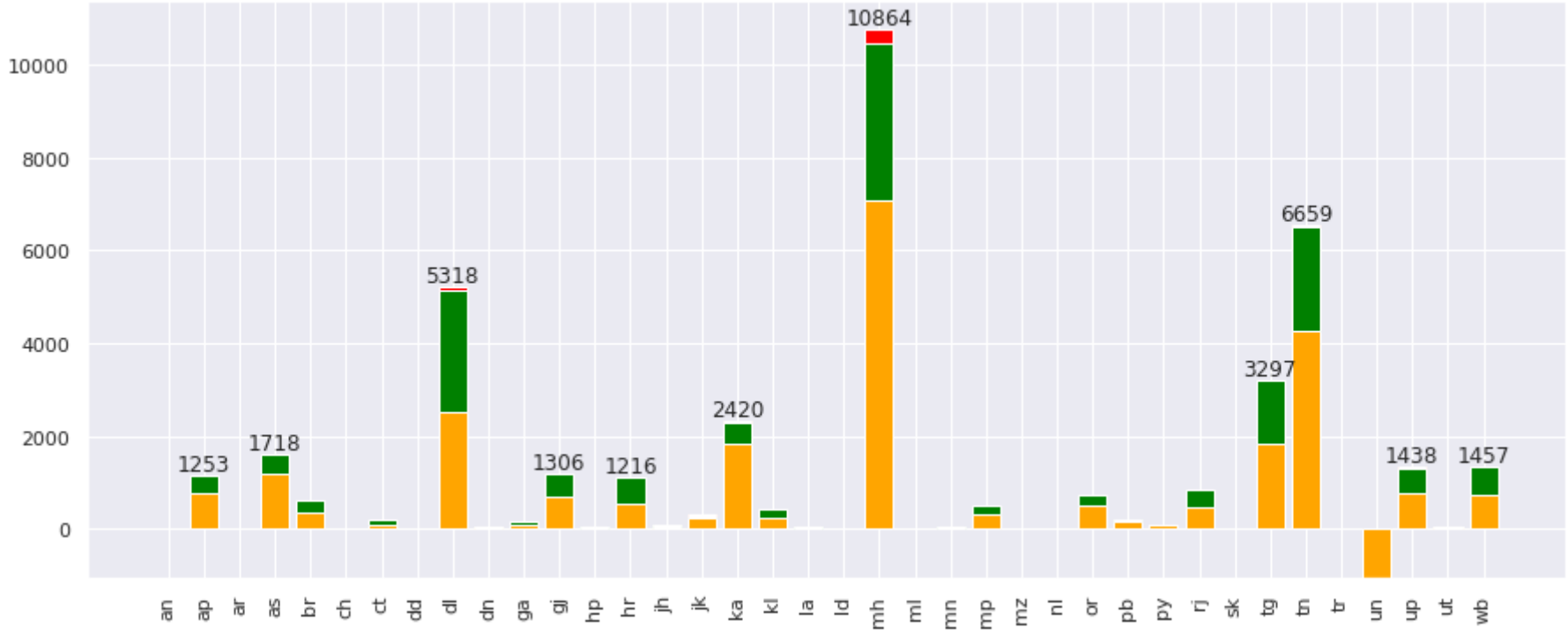
```
# setting the colour map
cmap = plt.get_cmap('tab20b')
outer_colors = cmap(np.array([0, 4, 8]))
inner_colors = cmap(np.array([1, 2, 3, 5, 6, 7, 9, 10, 11]))
# outer ring
plt.pie(c.sum(axis=1), labels=c.index,
        radius = 1, wedgeprops=dict(width=0.3),
        colors=outer_colors);
#inner ring
plt.pie(c.values.flatten(), radius=0.7,
        labels = ['A', '', 'G', 'A', 'C', '', 'A', '', ''],
        wedgeprops=dict(width=0.3),
        colors=inner_colors,
        labeldistance=0.75, textprops=dict(color='w'));
```



Stacked bar plots

1. Example using covid data - using last three rows
    - Seaborn does not allow stacked bar plots thus, matplotlib.pyplt is used.
    - The figure parameters can be set using plt.gcf() (an alternative to plt.subplots()) figure handle.
    - The bars can be annotated using plt.text().

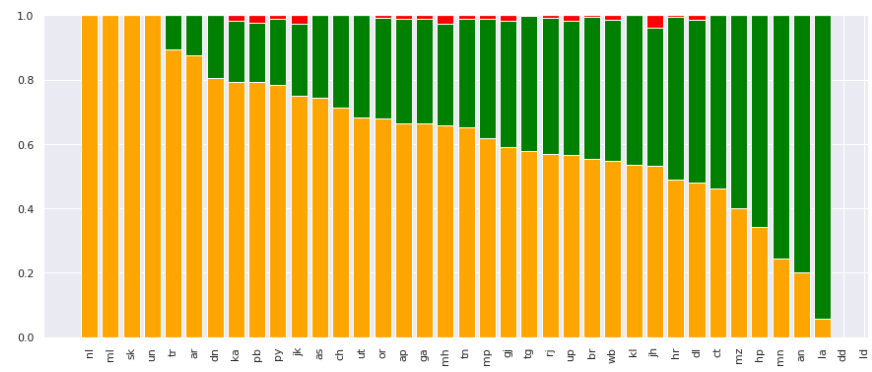
```
# pre-processing covid data for representation
df_ = df.tail(3)
df_.drop('date', axis=1, inplace=True)
df_.set_index('status', inplace=True)
df_ = df_.T
df_ = df_.apply(pd.to_numeric)
df_.drop('tt', inplace=True)

# defining the stacked bar plot
fig = plt.gcf();
fig.set_size_inches(15, 6);
plt.bar(df_.index, df_.Confirmed, color='Orange');
plt.bar(df_.index, df_.Recovered, bottom=df_.Confirmed, color='Green');
plt.bar(df_.index, df_.Deceased, bottom=df_.Confirmed + df_.Recovered, color='Red');
plt.xticks(rotation=90);

# adding the annotations
for i, val in enumerate(df_.index):
    y = df_.loc[val].sum() + 100
    if y > 1000:
        x = i
        plt.text(x, y, str(y), ha="center");
```



Relative stacked bar plot

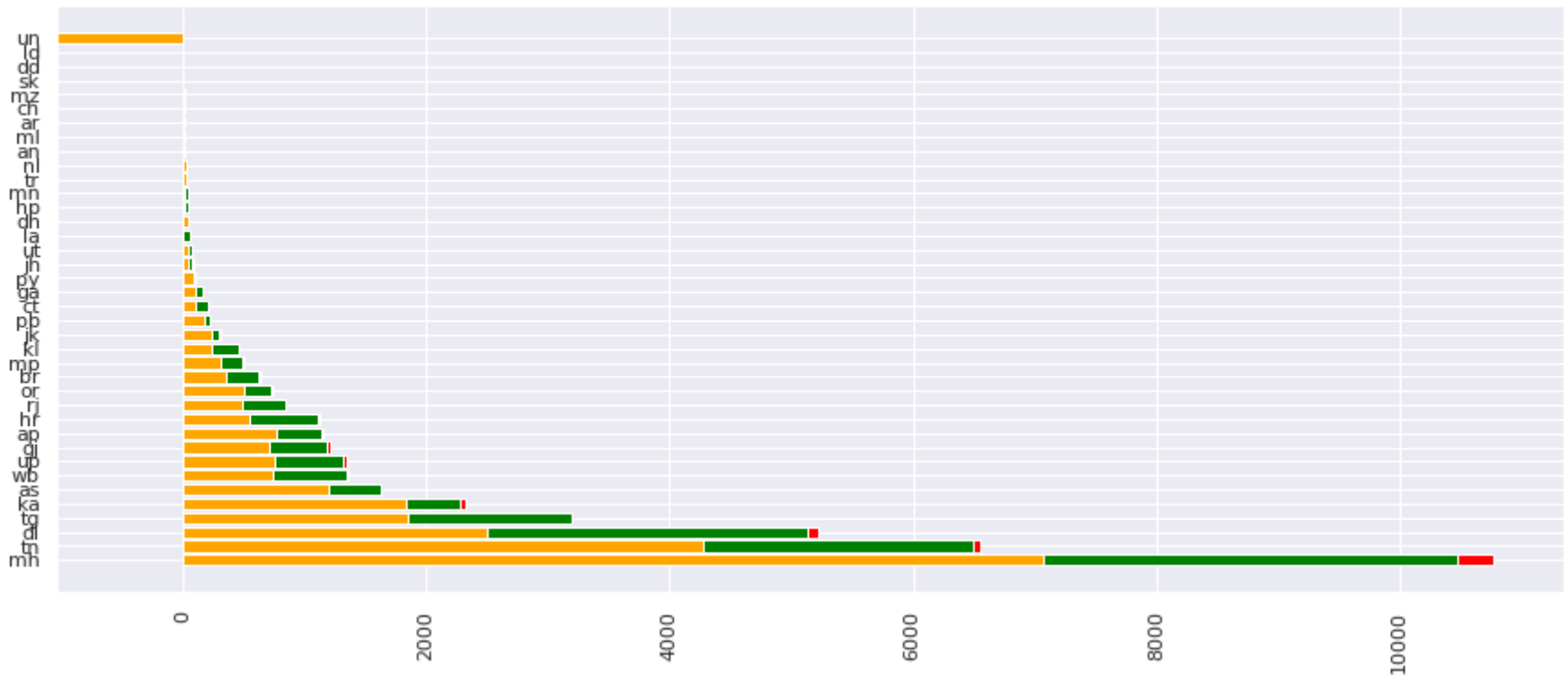
- The plots in the stacked bar plots are helpful to compare the cases in the given state not interstate comparison. For this the plots have to be relative.
- The addition of three resulting fractions add to one, therefore, the heights of all bars will be the same.
- Sorting the values aids in easier data comparison.

```
# preprocessing the covid data to include relative  fraction
df_['Total'] = df_.sum(axis = 1)
df_['ConfirmedFraction'] = df_['Confirmed'] / df_['Total']
df_['RecoveredFraction'] = df_['Recovered'] / df_['Total']
df_['DeceasedFraction'] = df_['Deceased'] / df_['Total']
# sorting by confirmed fraction
df_ = df_.sort_values('ConfirmedFraction', ascending=False)
# plotting relative stacked plot
fig = plt.gcf();
fig.set_size_inches(15, 6);
plt.bar(df_.index, df_.ConfirmedFraction, color='Orange');
plt.bar(df_.index, df_.RecoveredFraction, bottom=df_.ConfirmedFraction, color='Green');
plt.bar(df_.index, df_.DeceasedFraction, bottom=df_.ConfirmedFraction + df_.RecoveredFraction, color='Red');
plt.xticks(rotation=90);
```



2. Plotting stacked bar charts can be done horizontally, in descending order of total cases.

```
# sorting based on total cases
df_ = df_.sort_values('Total', ascending=False)
# plotting the bar plot
fig = plt.gcf();
fig.set_size_inches(15, 6);
plt.barh(df_.index, df_.Confirmed, color='Orange');
plt.barh(df_.index, df_.Recovered, left=df_.Confirmed, color='Green');
plt.barh(df_.index, df_.Deceased, left=df_.Confirmed + df_.Recovered, color='Red');
plt.xticks(rotation=90);
```



Time - varying composition of data

- The data should have date (of data type datetype) and pivot the data along the status axis.
- The required transformations on the data are shown using the covid data for ‘mh’.

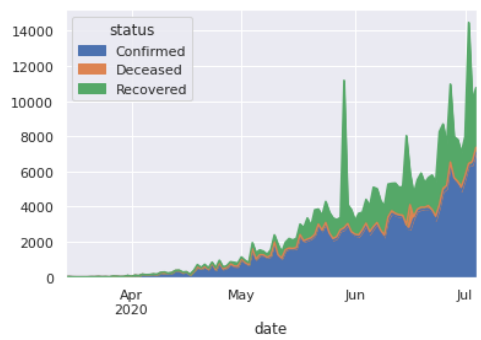
```
# filtering the required data
df_ = df[['mh', 'date', 'status']]
# converting the data types
df_['mh'] = pd.to_numeric(df_['mh'])
df_['date'] = pd.to_datetime(df_['date'])
# pivoting the data along the status axis
df_ = df_.pivot_table(values="mh", columns="status", index="date")
```

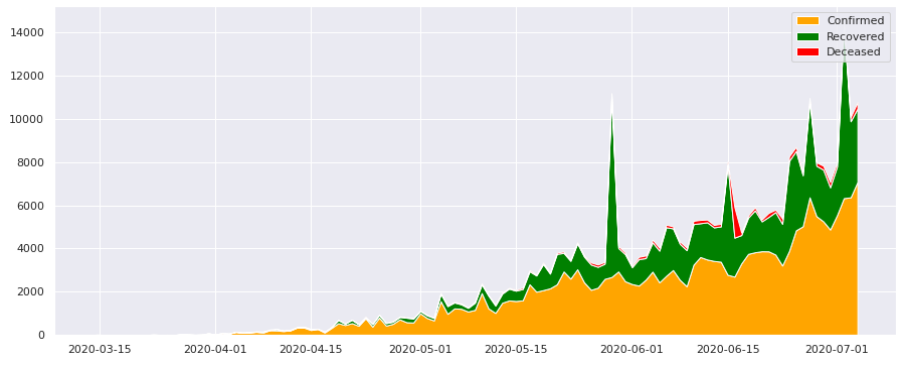
Stacked area plot

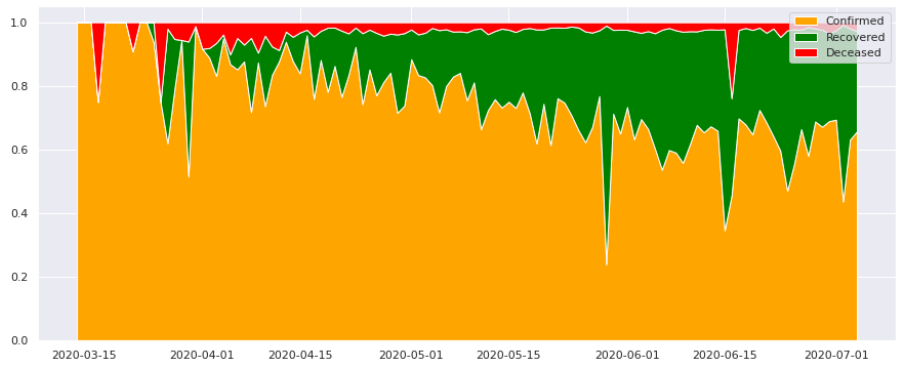
- Composition of data across time can be viewed.

1. Example plot using Maharashtra data

```
# first plot - using pandas
df_.plot.area();
# second plot - using matplotlib
fig = plt.gcf();
fig.set_size_inches(15, 6);
plt.stackplot(df_.index, df_.Confirmed, df_.Recovered, df_.Deceased,
              labels=['Confirmed', 'Recovered', 'Deceased'],
              colors=['orange', 'green', 'red']);
plt.legend();
# third plot - relative plot
fig = plt.gcf();
fig.set_size_inches(15, 6);
plt.stackplot(df_.index, df_.Confirmed/df_.sum(axis=1),
              df_.Recovered/df_.sum(axis=1),
              df_.Deceased/df_.sum(axis=1),
              labels=['Confirmed', 'Recovered', 'Deceased'],
              colors=['orange', 'green', 'red']);
plt.legend();
```







2. Writing a function to plot for a given state

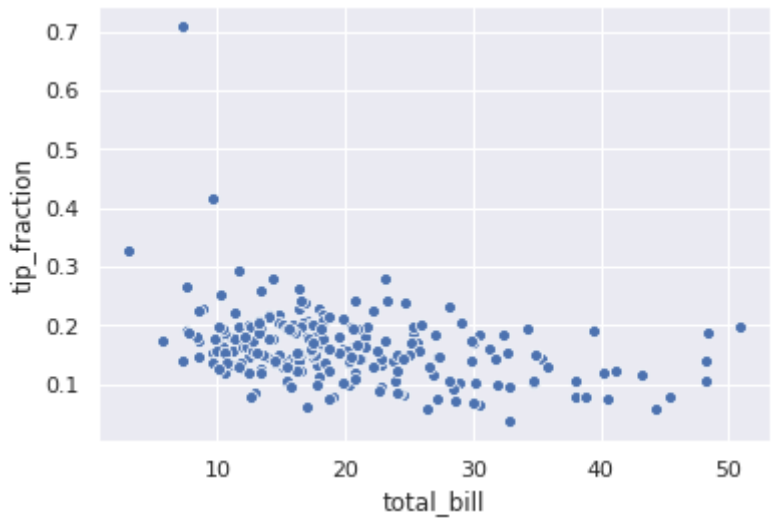
```
def plot_stacked_area_by_state(state):
    df_ = df[[state, 'date', 'status']]
    df_[state] = pd.to_numeric(df_[state])
    df_['date'] = pd.to_datetime(df_['date'])
    df_ = df_.pivot_table(values=state, columns="status", index="date")
    fig = plt.gcf();
    fig.set_size_inches(15, 6);
    plt.stackplot(df_.index, df_.Confirmed/df_.sum(axis=1),
                  df_.Recovered/df_.sum(axis=1),
                  df_.Deceased/df_.sum(axis=1),
                  labels=['Confirmed', 'Recovered', 'Deceased'],
                  colors=['orange', 'green', 'red']);
    plt.legend();

plot_stacked_area_by_state('tn')
```

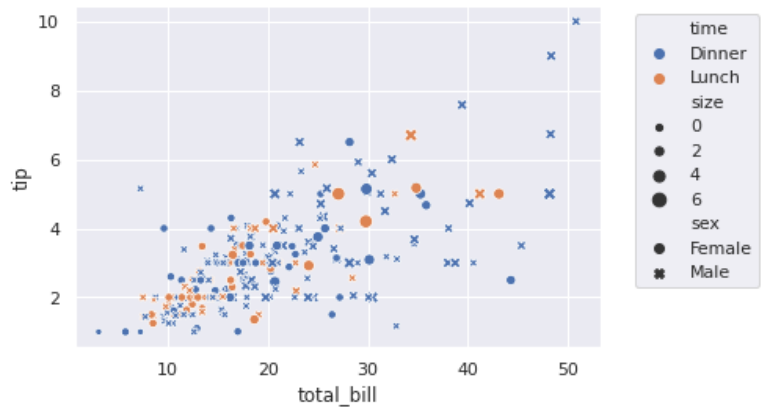
Scatter plots

- Used for plotting relationships between data, using seaborn. Scatterplots can be used for visualizing the distribution of variables as seen in the previous week.
- As seen earlier hue can be used to add an additional dimension to the plot.
- Additionally style and size can also be used.
- The position of the legend can be changed relative to x-axis by taking

1. Example using tips data

```
# loading dataset
t = sns.load_dataset('tips')
# first plot - setting hue for ordinal categorical data
sns.scatterplot(x='total_bill', y='tip', data=t,
                hue='size');
# second plot - relative fraction
t['tip_fraction'] = t['tip']/t['total_bill']
sns.scatterplot(x='total_bill', y='tip_fraction', data=t);
# third plot - setting hue for nominal data
sns.scatterplot(x='total_bill', y='tip', data=t,
                hue='time', style='sex', size='size');
plt.legend(bbox_to_anchor=(1.05, 1));
```

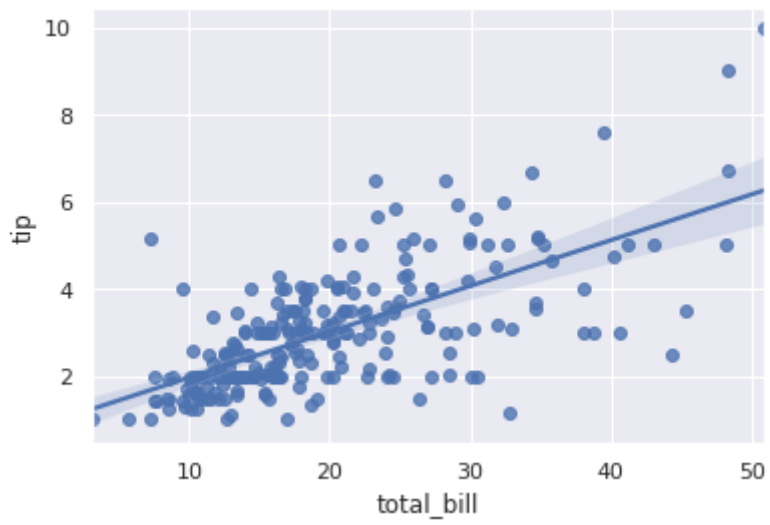


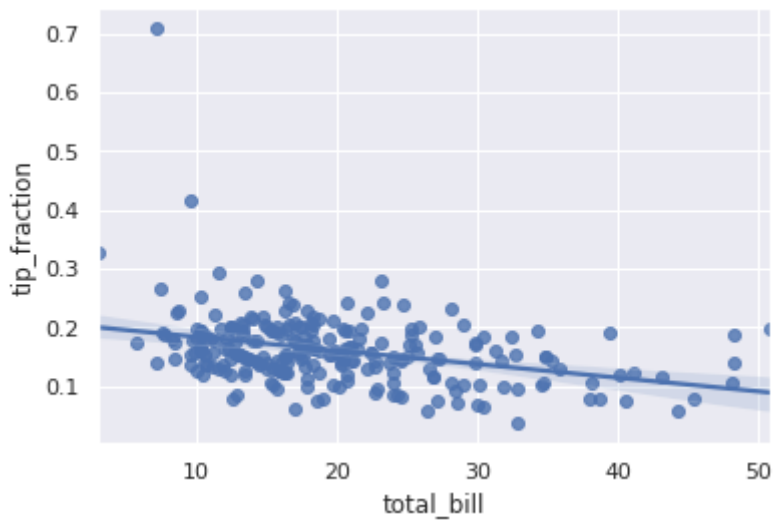


2. Using regression plots to analyse trends.

- The shaded region around the line depicts the error for the line around the 2D space. Closer the points to the line, lesser the error. As the points go farther away from the line, the error increases.
- The downward trend in the tips fraction can be seen in the second plot, indicating that expensive meals mean higher tips but it is a smaller fraction of the total bill amount.

```
# first plot
sns.regplot(x='total_bill', y='tip', data=t);
# second plot
sns.regplot(x='total_bill', y='tip_fraction', data=t);
```

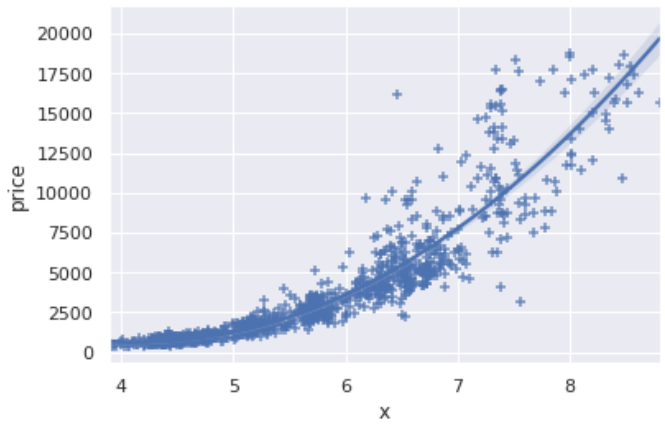




3. Example using diamonds dataset

- The order of the curve can be specified , here, as 2 for quadratic.

```
sns.regplot('x', 'price', data=d.sample(1000), order=2, marker="+");
```



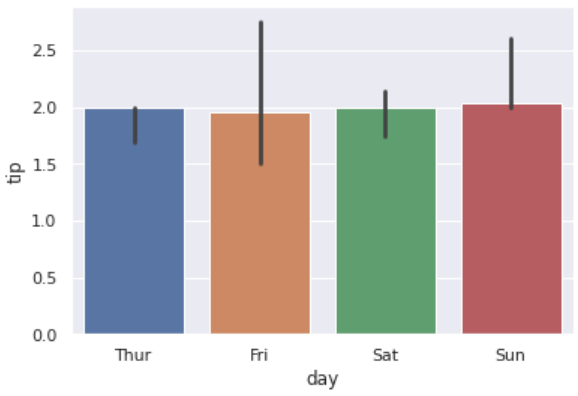
Bar plot

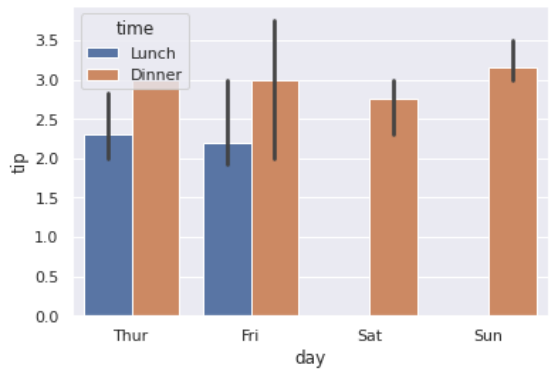
- For categorical variables, bar plots are used see relationships between continuous variable and a categorical variable.
- The height of the bar corresponds to the mean of the data points (continuous variable) corresponding to the categorical variable.

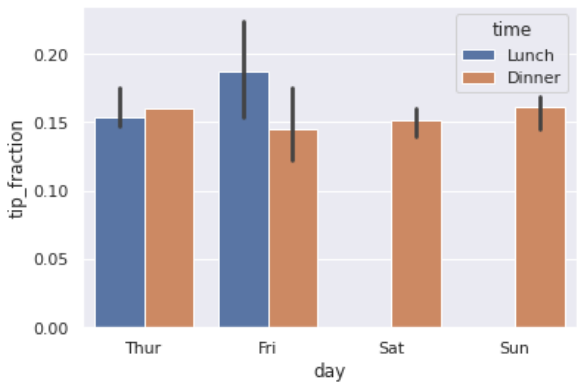
- The error bar corresponds to the confidence on the mean.
- The basic pandas functions of groupby, function and aggregation of results are done to plot the bar graph.
- The estimator parameter can be used to set the statistic (mean, median or self defined functions) .
- Hue can be used to add an additional dimension to the plot.

1. Example using the tips dataset

```
# first plot : user defined estimator
def my_estimate(v):
    return np.quantile(v, 0.25)
sns.barplot(x="day", y ="tip", data=t, estimator=my_estimate);
# second plot : adding hue time
sns.barplot(x="day", y ="tip", hue="time", data=t, estimator=np.median);
# third plot : time hue for tip_fraction
sns.barplot(x="day", y ="tip_fraction", hue="time", data=t, estimator=np.median);
```







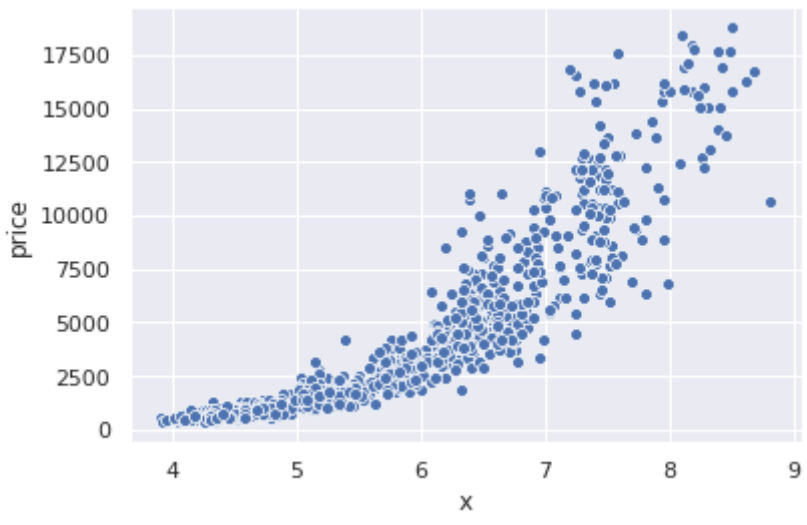
- From the second and third plot, it can be seen that the dinner bills are more expensive than the lunch.

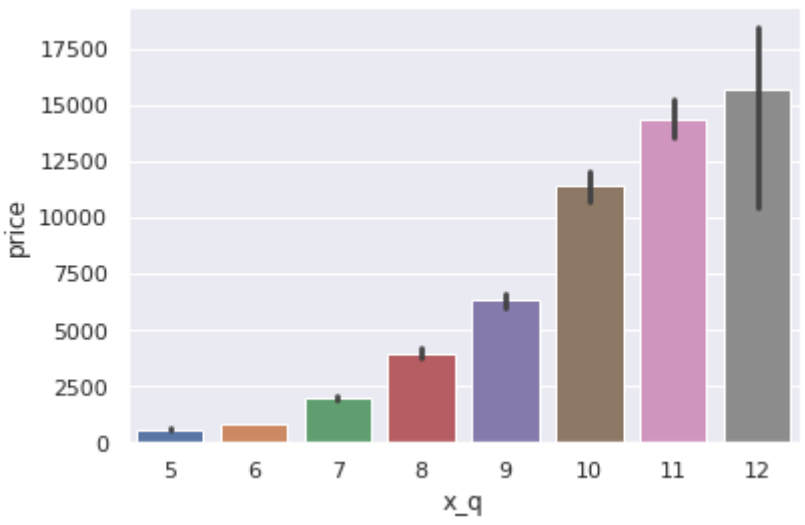
Continuous vs continuous plot

- In scatter plots cont. Y and cont. X are plotted where as in bar graphs a cont. Y is plotted against a discrete X variable.
- To convert continuous vs continuous plot (scatter plots) to cont. Vs discrete plot (bar plot), the continuous X is binned. This implies the intervals are defined and the membership of each X value is determined.

1. Example using diamonds dataset, the cont. variable is binned using pd.cut()

```
# first plot : scatter plot
sns.scatterplot('x', 'price', data=d.sample(1000));
# second plot : bar plot ( binned values)
d['x_q'] = pd.cut(d['x'], bins=15, labels=False);
sns.barplot('x_q', 'price', data=d.sample(1000));
```





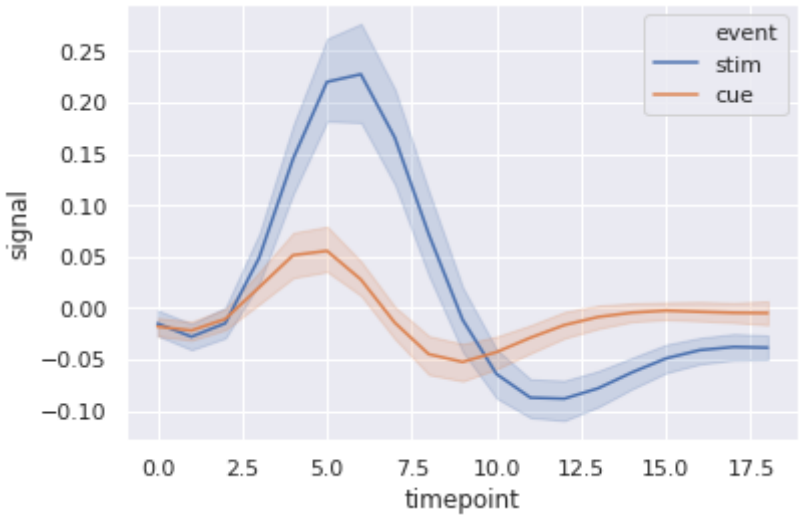
Line plot

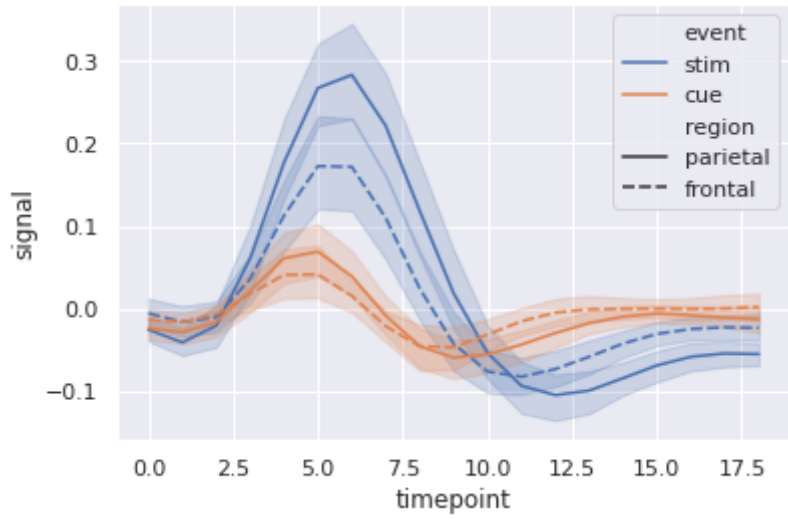
- It is a continuous vs continuous plot. Unlike scatter plot, instead of each point a trend line is plotted.

1. Example using fmri data

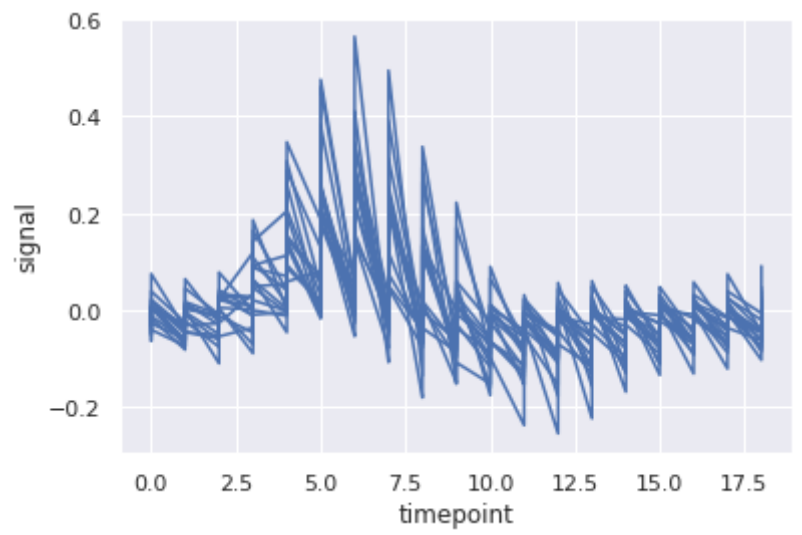
- The aggregated trend line is plotting along with its variability across the timeline.
- Additional dimensions can be added using hue, style.
- Markers can be set to view the points plotted. The estimator is mean by default and this can be changed using the estimator parameter.
- Units parameter along with the estimator set to None can be used to plot a trend line for each unique ‘unit’ parameter.

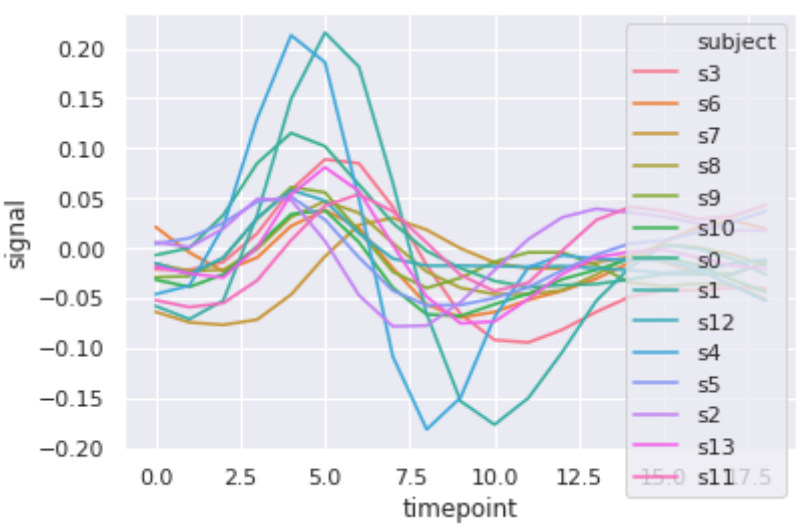
```
# first plot : time vs signal - aggregated w.r.t timepoint
f = sns.load_dataset('fmri')
sns.lineplot('timepoint', 'signal', data=f, hue="event");
# second plot : example using style
sns.lineplot('timepoint', 'signal', data=f, hue="event", style="region");
```



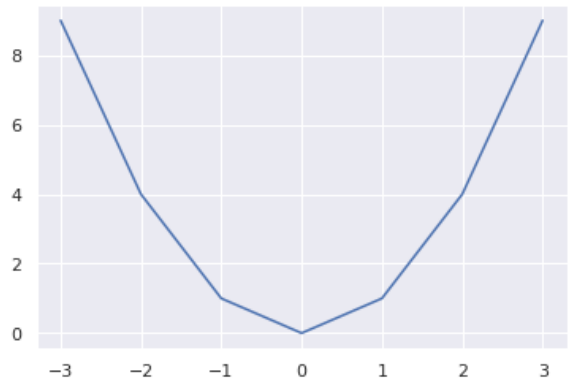



```
# plotting individually for a given categorical column
# first plot
sns.lineplot('timepoint', 'signal', data=f, units='subject', estimator=None);
# second plot
f_ = f[(f.region == "parietal") & (f.event == "cue")]
sns.lineplot('timepoint', 'signal', data=f_, hue='subject', estimator=None);
```






```
# example using numpy array
x = np.array([-3, -2, -1, 0, 1, 2, 3])
y = x * x
sns.lineplot(x, y);
```

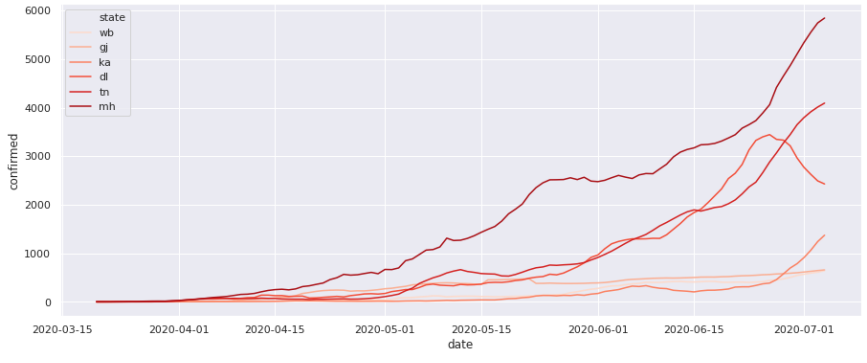


Line plot covid data

1. Plotting line plots across time for number of confirmed people across each state.

- A rolling average is calculated to account for the noise in the data.

```
# data pre-processing, melting is essentially reshaping the data
with open('data.json') as f:
  data = json.load(f)
data = data['states_daily']
df = pd.json_normalize(data)
df['date'] = pd.to_datetime(df['date'])
df.drop('tt', axis=1, inplace=True)
df.set_index('date', inplace=True)
df = df[df['status'] == 'Confirmed']
df.drop('status', axis=1, inplace=True)
df = df.apply(pd.to_numeric)
df = df.rolling(7).mean()
df.reset_index(inplace=True)
# plotting
fig = plt.gcf();
fig.set_size_inches(15, 6);
sns.lineplot('date', 'confirmed', hue="state", data=df_,
             palette='Reds', hue_order = ['wb', 'gj', 'ka', 'dl', 'tn', 'mh']);
```



Heat map

- Used to plot discrete variables on both x and y axes.
- The colour intensity is used to denote the corresponding numeric value.

1. Example using flights dataset

- From the dataset it can be inferred that across the years the trend is increasing. It is particularly large in the months of june to august.
- The data can be annotated with the numbers.
- The cmap can be set using diverging_palette, specifying the range and number of colours. The center value can be set and other colours are shaded with respect to this point as the zero point.

```
# leading dataset
fl = sns.load_dataset('flights')
# reshaping using pivot function
f1_ =  f1_.pivot( index = 'year' , columns = 'month', values = 'passengers')
# first plot : plotting
fig = plt.gcf();
fig.set_size_inches(15, 10)
sns.heatmap(fl_.T, annot=True, fmt="d", cmap="YlGnBu");
# second plot
fig = plt.gcf();
fig.set_size_inches(15, 10)
sns.heatmap(fl_.T, annot=True, fmt="d",
```

```
            cmap=sns.diverging_palette(250, 10, n=45),
            center=fl_.loc[1954, 'January']);
```